CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Cycle-Based Evolutionary Algorithm for the Fixed-Charge Capacitated Multi-Commodity Network Design Problem

Dimitris C. Paraskevopoulos
Tolga Bektas
Teodor Gabriel Crainic
Chris N. Potts

July 2014

CIRRELT-2014-36

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Université Laval · McGill · Concordia · ÉTS · UQÀM · HEC Montréal · Polytechnique Montréal · Université de Montréal

A Cycle-Based Evolutionary Algorithm for the Fixed-Charge Capacitated Multi-Commodity Network Design Problem†

Dimitris C. Paraskevopoulos[1], Tolga Bektaş[2], Teodor Gabriel Crainic[3,*], Chris N. Potts[2]

[1] School of Management, University of Bath, Claverton Down, Bath, BA2 7AY, United Kingdom

[2] School of Management,Centre for Operational Research, Management Science and Information systems (CORMSIS), University of Southampton, Southampton, SO17 1BJ, United Kingdom

[3] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Management and Technology, Université du Québec à Montréal, P.O. Box 8888, Station Centre-Ville, Montréal, Canada H3C 3P8

Abstract. This paper presents an evolutionary algorithm for the fixed-charge multicommodity network design problem (MCNDP), which concerns routing multiple commodities from origins to destinations by designing a network through selecting arcs, with an objective of minimizing the fixed costs of the selected arcs plus the variable costs of the flows on each arc. The proposed algorithm evolves a pool of solutions using principles of scatter search, interlinked with an iterated local search as an improvement method. New cycle-based neighbourhood operators are presented which enable complete or partial re-routing of multiple commodities. An efficient perturbation strategy, inspired by ejection chains, is introduced to perform local compound cycle-based moves to explore different parts of the solution space. The algorithm also allows infeasible solutions violating arc capacities while forming the "ejection cycles", and subsequently restores feasibility by systematically applying correction moves. Computational experiments on benchmark MCNDP instances show that the proposed solution method consistently produces high-quality solutions in reasonable computational times.

Keywords: Multi-commodity network design, scatter search, evolutionary algorithms, ejection chains, iterated local search.

Acknowledgements. The authors gratefully acknowledge the following sources of funding for this project: the Engineering and Physical Sciences Research Council (EPSRC), the Rail Safety and Standards Board (RSSB) and Department of Transport, UK, the Natural Sciences and Engineering Research Council of Canada (NSERC) through its Discovery Grant programs, the Faculty Strategic Research Fund provided by the Faculty of Business and Law at the University of Southampton, and the Fonds de recherche du Québec, Canada for their infrastructure grants.

†Revised version of the CIRRELT-2013-08.

Results and views expressed in this publication are the sole responsibility of the authors and do not necessarily reflect those of CIRRELT.

Les résultats et opinions contenus dans cette publication ne reflètent pas nécessairement la position du CIRRELT et n'engagent pas sa responsabilité.

---

* Corresponding author: TeodorGabriel.Crainic@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2014

© Paraskevopoulos, Bektaş, Crainic, Potts and CIRRELT, 2014

1 Introduction

The fixed-charge capacitated multi-commodity network design problem (MCNDP) consists of designing a network on a given graph by selecting arcs to route a given set of commodities between origin-destination pairs. Each arc has a predefined capacity specifying the maximum flow that the arc can accommodate. Also, associated with each arc are fixed and variable costs, where the fixed cost is incurred only if the arc is selected, and the variable cost is a cost per unit of flow along the arc. Each commodity is defined by an origin and a destination node and the amount to be transported. The objective is to minimize the total cost of establishing the arcs and routing the flows.

The MCNDP has attracted much attention in the literature due to both its complexity (the problem is NP-hard in the strong sense), and a wide variety of applications in the areas of telecommunications, logistics, production and transportation systems (Balakrishnan et al., 1997; Magnanti and Wong, 1986; Minoux, 1986). Despite the significant efforts devoted to the development of exact methodologies for the MCNDP (Crainic et al., 2001; Hewitt et al., 2010), the literature still favours heuristic approaches when large-scale problem instances are involved. One of the most successful local search strategies for the MCNDP is proposed by Ghamlouche et al. (2003), where new cycle-based neighbourhood operators are incorporated in a tabu search framework. The cycle-based operators are subsequently used within a path-relinking algorithm (Ghamlouche et al., 2004), a multilevel cooperative framework (Crainic et al., 2006), and a scatter search (SS) (Crainic and Gendreau, 2007). In the latter paper, the authors conclude that the proposed SS failed to meet their expectations and further research is needed to realize the full potential of SS.

Inspired and motivated by the advances in the heuristic approaches for the MCNDP, this paper contributes to the existing body of work by: (i) proposing an efficient iterated local search (ILS) that utilizes new and enhanced cycle-based neighbourhood operators, long and short term memory structures, and an innovative perturbation strategy based on ejection chains (Glover, 1996) that aims at guiding the search towards unexplored regions of the solution space; (ii) introducing an efficient SS that considers the search history and "solvency-based" measures to produce offspring; (iii) presenting results of computational experiments conducted on benchmark instances using an algorithm incorporating the various elements described above. The majority of the heuristics for the MCNDP utilize a trajectory-based or an evolutionary framework to select arcs for inclusion in the design, and subsequently call a commercial optimizer (e.g., CPLEX) to solve the corresponding flow subproblem. As the flow subproblems become larger, the solution time for repeatedly finding minimum cost flows might become significant, even though linear programming optimizers are relatively efficient. Towards this end, we call the linear programming (LP) solver as few times as possible in the proposed algorithm in order to reduce its computational requirements.

The remainder of this paper is organized as follows. Section 2 provides a brief review of the recent literature on the MCNDP. Section 3 presents our evolutionary algorithm and all of its components, namely the initialization phase, the SS, and the ILS. In Section 4, we describe details of our computational experiments, and we also present results of applying the proposed algorithm to benchmark MCNDP instances from the literature. Conclusions are given in Section 5, where future research directions are also presented.

2 Literature

A number of efficient algorithms have appeared in the literature to address the inherent complexity of solving the MCNDP. In this section, we provide a brief review of the available methods but focus on heuristic, as opposed to exact, solution algorithms for reasons stated earlier.

Crainic et al. (2000) propose a simplex-based tabu search method for the MCNDP using a path-flow based formulation of the problem. Their method combines column generation with pivot-like moves of single commodity flows to define the path flow variables. In a similar fashion, Ghamlouche et al. (2003) describe cycle-based neighbourhoods for use in metaheuristics aimed at solving MCNDPs. The main idea of the cycle-based local moves is to redirect commodity flows around cycles in order to remove existing arcs from the network and replace them with new arcs. They use the proposed neighbourhood structures in a tabu search algorithm, where a commodity flow subproblem is solved to optimality at each iteration.

Ghamlouche et al. (2004) propose an evolutionary algorithm for the MCNDP. Their solution framework is based on path relinking, in which cycle-based neighbourhoods are used to generate an elite candidate set of solutions in a tabu search algorithm and for moving from the initial to the guiding solution. When updating the pool of solutions, the dissimilarity of solutions is considered as an additional component in calculating the solution value. Alvarez et al. (2005) describe an SS algorithm for the MCNDP. They use GRASP, originally proposed by Feo and Resende (1995), to produce a diversified initial set of solutions. Each commodity path is subject to an improvement process. The solutions are combined by choosing the best path for each commodity among the solutions that are being combined. A feasibility restoration mechanism is also available for solutions that are infeasible. In contrast to the recombination process of Alvarez et al. (2005), our SS does not consider commodity paths to build a solution; instead, independent arcs are combined to create offspring. We believe that the latter enhances the SS algorithm's capabilities, as more combinations can occur when arcs instead of paths are combined together, leading to a rich pool of offspring.

A parallel cooperative strategy is described by Crainic and Gendreau (2002) using

tabu search and various communication strategies. In a similar fashion, Crainic et al. (2006) propose a multilevel cooperative search on the basis of local interactions among cooperative searches and controlled information gathering and diffusion. The focus of their algorithm is on the specification of the problem instance solved at each level and the definition of the cooperation operators.

Katayama et al. (2009) propose a column and row generation heuristic for solving the MCNDP. The authors relax the arcs' capacity constraints, while a column and row generation technique is developed to solve the relaxed problem. Using similar ideas, Yaghini et al. (2013) present a hybrid simulated annealing (SA) and column generation (CG) algorithm for solving the MCNDP. The SA is used to define the open and closed arcs, wherein the flow subproblem is solved via CG.

A local branching technique for the MCNDP is proposed by Rodríguez-Martín and Salazar-González (2010). Even though the method, originally proposed by Fischetti and Lodi (2003), is exact by nature, high quality heuristic solutions can be produced using an MIP solver as a "black box". A solution framework that employs a combination of mathematical programming algorithms and heuristic search techniques is introduced by Hewitt et al. (2010). Their methodology uses very large neighbourhood search in combination with an IP solver on an arc-based formulation of the MCNDP, and a linear programming relaxation of the path-based formulation using cuts discovered during the neighbourhood search. A follow-up study by Hewitt et al. (2012) introduces a generic branch-and-price guided algorithm for integer programs with an application to the MCNDP.

3 Solution Methodology

In this section, we first present a formal definition of the problem including the notation that will be used in the rest of the paper and then describe in detail the components of the main algorithm.

3.1 Problem definition

The MCNDP is defined on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. Each arc $(i, j) \in \mathcal{A}$ has an associated fixed cost f_{ij} that is incurred if it is selected for inclusion in the network, has a cost per unit of flow c_{ij}, and has a capacity u_{ij}. A set of commodities denoted by $\mathcal{P}$ is given, where each commodity has an origin, a destination, and a quantity to be shipped from origin to destination. Problems with more than one origin or destination per commodity can be modelled by splitting commodities (see Holmberg and Yuan, 2000).

The goal of the problem is to select a subset of arcs that are to be included in the final design of the network along with the commodity flows on these arcs, to minimize the total cost of the selected arcs and the flow distribution on the resulting network. For simplicity, we will refer to the arcs that are included in the final design of the network as *open* arcs; otherwise, the arcs should be considered as *closed*. Binary variables y_{ij} are used, where $y_{ij} = 1$ if the arc $(i,j) \in \mathcal{A}$ is open, and $y_{ij} = 0$ otherwise. The flow on each arc $(i,j) \in \mathcal{A}$ that is used for shipping each commodity $p \in \mathcal{P}$ from its origin to its destination is denoted by x_{ij}^p. Conservation of flow constraints must be satisfied at each node, and there are capacity constraints of the form $\sum_{p \in \mathcal{P}} x_{ij}^p \leq u_{ij}$ for each $(i,j) \in \mathcal{A}$. The cost $f(s)$ of a solution s that is defined by variables x_{ij}^p and y_{ij} for $(i,j) \in \mathcal{A}$ and $p \in \mathcal{P}$ is computed using

$$f(s) = \sum_{(i,j) \in \mathcal{A}} \sum_{p \in \mathcal{P}} c_{ij} x_{ij}^p + \sum_{(i,j) \in \mathcal{A}} f_{ij} y_{ij}. \tag{1}$$

We adopt the convention that $f(s) = \infty$ if solution s is infeasible.

Two types of mathematical formulations for the problem appear in the literature; an arc-based and a path-based formulation. We refer to Gendron et al. (1998), Frangioni and Gendron (2001) and Hewitt et al. (2010) for details of these mathematical formulations.

3.2 Evolutionary algorithm

Our proposed solution methodology is an evolutionary algorithm that evolves a population of solutions using the principles of SS and applies ILS (Lourenço et al., 2002) as an improvement method. Following the basic template of the SS framework, our solution approach is composed of three distinct phases: (i) an *Initialization phase* where a population of good and diverse solutions is produced and a Reference Set (set R) is initialized; (ii) a *Scatter Search phase* where a recombination process takes place to produce offspring; and (iii) an *Education phase* where these offspring (hosted in set C) are "educated" by attempting to improve their quality via the proposed ILS. Computational time is used as the termination criterion. The framework is given in Algorithm 1.

Algorithm 1: Evolutionary Algorithm

Input: λ (initial population size), μ (Reference Set size), where $\lambda \geq \mu$, δ (number of local search iterations without an improvement), κ (Candidate Set size), $\vartheta_{\max}$ (number of LP solver calls within local search without an improvement)

Output: R, $s_{\text{best}} \in R$

1. Initialization phase

$R \leftarrow \text{ConstructionHeur}(\lambda, \mu)$;

while *termination conditions* **do**

  2. Scatter Search phase

  $C \leftarrow \text{SolutionCombination}(\kappa, \mu)$;

  3. Education phase

  for *individual* s *of* C **do**

    $s' \leftarrow \text{ILS}(s, \delta, \vartheta_{\max})$;

    $\text{UpdateRefSet}(R, s')$;

Details of the three phases of Algorithm 1 are explained in the subsections below. Prior to this, however, we describe a flow routing procedure that is used in each phase of our algorithm.

3.3 Routing/re-routing procedure

In the Initialization phase of Algorithm 1, solutions are created by successively adding flow to an existing partial solution by selecting a commodity and routing its required flow from origin to destination. A similar solution creation method is used in some iterations of the Scatter Search phase where some open arcs are selected by the solution recombination method but more are needed to create a feasible flow. Finally, when applying Iterated Local Search within the Education phase, re-routing of flow is applied both in the process for creating neighbours of the current solution and in the procedure for perturbing the current solution. In each of these phases, the routing or re-routing is determined from the solution of a shortest path problem that is obtained by applying Dijkstra's algorithm. We now provide details of how these shortest path problems are defined.

Consider a partial solution defined by $y_{ij} = \bar{y}_{ij}$ and $x^p_{ij} = \bar{x}^p_{ij}$ for each arc $(i,j) \in \mathcal{A}$ and each commodity $p \in \mathcal{P}$. Thus, $\bar{y}_{ij} = 1$ for each arc (i,j) that is open in the partial solution, $u_{ij} - \sum_{p \in \mathcal{P}} \bar{x}^p_{ij}$ is the remaining capacity in each arc (i,j). The aim is to route w^p units of flow of some commodity p from node i^p to node j^p, for appropriately defined w^p and nodes i^p and j^p.

There are two alternative shortest path problems that we define. For a *single-path routing*, a path from i^p to j^p is required such that each of arc of the path has a remaining capacity of at least w^p, thereby allowing all of the desired w^p units of flow to be routed along this path. On the other hand, for *multiple-path routing*, it is sufficient to find a path where each of its arcs has a non-zero remaining capacity. For both types of routing, we define a shortest path problem on the original graph $\mathcal{G}$ with a cost $\bar{c}_{ij}$ for each arc $(i,j) \in \mathcal{A}$, for suitably defined values of $\bar{c}_{ij}$.

For single-path routing, we define for each arc $(i,j) \in \mathcal{A}$

$$\bar{c}_{ij} = \begin{cases} w^p c_{ij} + f_{ij}(1 - \bar{y}_{ij}) & \text{if } w^p \leq u_{ij} - \sum_{p \in \mathcal{P}} \bar{x}^p_{ij}, \\ \infty & \text{otherwise.} \end{cases}$$

Thus, only arcs that can accommodate an additional w^p units of flow have a finite cost. For an arc (i,j) that can accommodate this additional flow, $\bar{c}_{ij}$ is the cost of that flow in arc (i,j) plus any additional cost of opening arc (i,j) if it is not already open in the current partial solution.

For multiple-path routing, we similarly define for each arc $(i,j) \in \mathcal{A}$

$$\bar{c}_{ij} = \begin{cases} \min\{u_{ij} - \sum_{p \in \mathcal{P}} x^p_{ij}, w^p\} c_{ij} + f_{ij}(1 - y_{ij}) & \text{if } \sum_{p \in \mathcal{P}} x^p_{ij} < u_{ij}, \\ \infty & \text{otherwise.} \end{cases}$$

In this case, arc (i,j) can be used for an additional $\min\{u_{ij} - \sum_{p \in \mathcal{P}} \bar{x}^p_{ij}, w^p\}$ units of flow. If this value is strictly positive, then $\bar{c}_{ij}$ is the is the cost of that flow in arc (i,j) plus any additional cost of opening arc (i,j); otherwise, arc (i,j) cannot accommodate any additional flow and therefore the value of $\bar{c}_{ij}$ is set to infinity.

For both types of routing, Dijkstra's algorithm is applied to find the shortest path from node i^p to node j^p. If the shortest path length is not finite, then no routing of flow is possible. Otherwise, in the case of single-path routing, a flow augmentation process adds w^p units of flow to all arcs of the shortest path from node i^p to node j^p. Analogously, in the case of multiple-path routing, if P is the shortest path from i^p to j^p, then $\min\{\min_{(i,j) \in P}\{u_{ij} - \sum_{p \in \mathcal{P}} \bar{x}^p_{ij}\}, w^p\}$ units of flow are added to all arcs of P.

3.4 Initialization phase

In the Initialization phase, each iteration of the construction heuristic selects an unrouted or partially routed commodity p at random. Then, a random choice is made as to whether a single-path or multiple-path routing is to be attempted with an equal probability for each choice.

For a single-path routing, w^p is amount of flow for commodity p that is to be routed, and i^p and j^p are the origin and destinations nodes for this flow. If the shortest path computation, as described in Section 3.3, provides a solution with a finite shortest path length, then the flow is augmented. After removing commodity p from the set of unrouted or partially routed commodities and updating the variables y_{ij} and x_{ij}^p, the heuristic proceeds to the next iteration. Otherwise, no single-path routing of the chosen commodity exists, and this iteration is repeated using multiple-path routing.

For a multiple-path routing, w^p, i^p and j^p are defined as above. Following the shortest path computation described in Section 3.3, the flow is augmented and the values of $\bar{w}^p$, y_{ij} and $\bar{x}_{ij}^p$ are updated (with commodity p removed from the list of unrouted or partially routed commodities if w^p is reduced to zero), and the heuristic proceeds to the next iteration.

The construction heuristic is applied repeatedly until λ different solutions are created, among which μ are selected to build the Reference Set. Details about the creation of the initial Reference Set are given in Section 3.5.1.

3.5 Scatter Search

The SS phase evolves the Reference Set of solutions using an efficient recombination method as follows. A subset generation method selects κ solutions from the Reference Set, which form the Candidate Set (CS), and a solution combination method is then applied to produce one solution. This procedure is repeated until 2μ offspring are produced, which is double the number of parent solutions in the Reference Set. We choose μ best solutions, in terms of the solution cost, out of the 2μ offspring to proceed to the next phase. Other strategies were also tested, such as randomly choosing μ of 2μ solutions, but the algorithm performs better by choosing μ best solutions. The offspring are checked as to whether they meet the criteria to be inserted into the Reference Set or not, before proceeding to the Education phase. In the Education phase, ILS is used to improve the quality of each offspring, before these offspring are checked again for insertion into the Reference Set according to elitist criteria. These procedures are explained further in the following subsections.

3.5.1 Reference Set

The goal of using a Reference Set R is to maintain a balance between quality and diversity of solutions, and to avoid a premature convergence of the algorithm. An obvious measure of the quality of a solution s is its cost $f(s)$. An alternative quality measure that becomes

relevant after the evolutionary process has started is the *Solvency Ratio*, defined by

$$\mathrm{SR}(s) = \mathrm{neo}(s)/\mathrm{hits}(s), \tag{2}$$

where $\mathrm{hits}(s)$ denotes the number of times that solution s has participated in the recombination process to produce an offspring, and $\mathrm{neo}(s)$ denotes the *number of educated offspring* of s, which is the number of times that an offspring of s has been educated and included in R. The smaller the Solvency Ratio, the lower the value of the particular solution s is to the evolution process. In this way, a higher cost solution with respect to the usual objective function f may be beneficial to the search if it produces well-educated offspring. Our diversity measure uses the Hamming distance between pairs of solutions, $D(s,s') = \sum_{(i,j)\in\mathcal{A}} |y_{ij}^{s} - y_{ij}^{s'}|$, for two solutions s and s'. The total dissimilarity for Reference Set R is then defined by

$$\mathrm{TD}(R) = \sum_{s,s'\in R} D(s,s'), \tag{3}$$

where the sum is over all $\mu(\mu-1)/2$ pairs of solutions in set R.

The creation of the initial Reference Set proceeds as follows. The first μ solutions among the λ generated within the Initialization phase are inserted into R. The remaining $\lambda-\mu$ solutions are then considered sequentially for replacing a solution in R. Specifically, if such a solution s satisfies the condition $f(s) < f(s_{\mathrm{best}})$, where s_{best} is a least cost solution in R, or if there is a solution $r \in R$ for which $f(s) < f(r)$ and $D(r, s_{\mathrm{best}}) < D(s, s_{\mathrm{best}})$, then s is inserted into R. Otherwise, s is not included in R. When s is inserted, solution $s_{\mathrm{worst}} \in R$, where s_{worst} has the largest cost among solutions in R, is removed from R.

At each SS iteration of the evolutionary process, 2μ offspring are generated, and a sequence of decisions is made on whether to replace a solution in R with the offspring under consideration. In the later stages of the evolutionary process, this decision depends on the value of the SR obtained from (2), but a different process is used at the start of the evolutionary process when SR cannot be meaningfully computed. Specifically, let $s_{\mathrm{cand}} \in R$ be the candidate for removal from the reference set R, where $s_{\mathrm{cand}} = s_{\mathrm{worst}}$ for the first two iterations of the evolutionary process, and s_{cand} is the solution in R having the smallest Solvency Ratio from the third iteration onwards. An offspring s replaces s_{cand} in the Reference Set R if either $f(s) < f(s_{\mathrm{best}})$, or if $f(s) < f(s_{\mathrm{cand}})$ and $\mathrm{TD}(R) < \mathrm{TD}(R \setminus \{s_{\mathrm{cand}}\} \cup \{s\})$. This procedure differs from other studies where the usual practice is always to remove the worst-cost solution s_{worst} from R without taking into account any effect it might have on the evolution.

3.5.2 Solution combination method

In this section, we discuss how our proposed solution combination method generates each offspring.

Each offspring is generated from the *candidate set* (CS) comprising κ solutions from the Reference Set R. The solutions in CS are chosen probabilistically with a bias towards promising parents as determined by their Solvency Ratios. Specifically, the probability of a solution s being included in the candidate set is proportional to $\mathrm{SR}(s)$. In this way, a solution s with a low $\mathrm{SR}(s)$ is gradually neglected, and the focus is on new solutions that produce well-educated offspring. Because the Solvency Ratio changes while SS iterations are being performed, the scatter search phase has a dynamic character, and premature convergence is typically averted. Furthermore, to enable diversification, a penalty (as expressed by the term $\alpha\mathrm{hits}(s)$ in equations (4) and (5) below) is used to weaken the impact of a frequently selected parent and thereby enable diversification.

The arcs of the solutions in CS are combined to produce an offspring. For a given solution s, each arc (i,j) is either open if $y^s_{ij} = 1$ or closed if $y^s_{ij} = 0$. We associate a value $f(s) + \alpha\mathrm{hits}(s)$ with solution s, where $f(s)$ and $\mathrm{hits}(s)$ are previously defined, and α is a scaling parameter. We now introduce a scoring procedure to determine whether an arc (i,j) will be open or closed in the offspring, according to the following scores:

$$\mathrm{Op}_{ij} = \sum_{s \in \mathrm{CS}} \frac{y^s_{ij}}{f(s) + \alpha\mathrm{hits}(s)} \qquad \forall (i,j) \in \mathcal{A} \tag{4}$$

$$\mathrm{Cl}_{ij} = \sum_{s \in \mathrm{CS}} \frac{1 - y^s_{ij}}{f(s) + \alpha\mathrm{hits}(s)} \qquad \forall (i,j) \in \mathcal{A}. \tag{5}$$

Op_{ij} and Cl_{ij} are the scores for arc (i,j) being open and closed, respectively, and if $\mathrm{Op}_{ij} > \mathrm{Cl}_{ij}$, the preferred arc status is open; otherwise its preferred status is closed.

The preferred status of open or closed for each arc (i,j) is our starting point for creating a new solution from the solutions in the CS. We first assume that the open and closed arcs correspond to their preferred status, which implies that the values of the y_{ij} variables are fixed. To determine values of the x^p_{ij} variables or conclude that there is no feasible solution with the fixed y_{ij} variables, the associated capacitated multicommodity network flow problem is solved using an LP optimizer. If a feasible solution is obtained, then this is the offspring obtained from the candidate set (but with any open arc having a zero flow having its status changed to closed).

If the multicommodity network flow problem is infeasible, then the offspring is created using similar methodology to that of the Initialization phase as described in Section 3.4. Specifically, for each arc (i,j) with a preferred status of open, we temporarily change the fixed cost to f_{ij}/M and the cost per unit of flow to c_{ij}/M, where M is a large constant.

With these updated costs, the construction heuristic is applied, and the resulting solution is the offspring obtained from the CS. The low costs associated with the arcs having a preferred status of open encourages Dijkstra's algorithm to find shortest paths containing some of these arcs, which results in a large proportion of such arcs being open in the offspring solution.

3.6 Education phase: the ILS heuristic

The μ elite offspring, chosen among the 2μ produced by the SS phase, are individually "educated" (i.e., improved) using ILS. The components of the ILS are shown in Algorithm 2.

Algorithm 2: Iterated Local Search

Input: s (current offspring), δ (number of local search iterations without an improvement), $\vartheta_{\max}$ (number of LP solver calls without an improvement)
Output: s_{ILSBest} (the best solution found by ILS)

```
ϑ = 0; h⃗ ← 1; s_ILSBest ← s;
while ϑ < ϑ_max do
    g⃗ ← 0; iter ← 0; s' ← s;
    while iter < δ do
        (s'', g⃗) ← NeighbourhoodSearch(s', g⃗);
        if f(s'') ≤ f(s) then
            s ← s''; g⃗ ← 0;
        else
            iter = iter + 1;
        s' ← s'';
    s' ← LPsolver(s);
    if f(s') ≥ f(s_ILSBest) then
        s* ← EjectionCycles(s', h⃗);
        s ← s*; ϑ = ϑ + 1;
    else
        ϑ = 0; s ← s'; s_ILSBest ← s';
```

The proposed ILS has two main components, namely a local search and a perturbation strategy. The proposed local search uses new neighbourhood operators and short term memory (represented by memory structure $\vec{g}$) to avoid cycling. The perturbation strategy, namely Ejection Cycles, partially modifies the current solution according to information gathered during the search (long-term memory depicted by $\vec{h}$) in the spirit of Ejection Chains (Glover 1996).

3.6.1 Neighbourhoods and moves

Our ILS neighbourhood is based on the cycle-based operator, as originally proposed by Ghamlouche et al. (2003). Their approach is to select a pair of nodes containing a positive flow and then re-route the flows of the individual commodities between these nodes. In this paper, we design a more efficient and effective approach based on the notion of inefficient arcs and inefficient chains, as described below. Further, we allow a partial re-routing of flow that maintains flow feasibility. In contrast, Ghamlouche et al. (2003) remove all flow between the two selected nodes, and if the new flows do not result in a feasible solution, then a feasibility restoring routine is applied.

Consider a solution defined by the variables x_{ij}^p and y_{ij} for each arc $(i,j) \in \mathcal{A}$ and each commodity $p \in \mathcal{P}$. For each open arc (i,j), where $y_{ij} = 1$ and $x_{ij}^p > 0$ for at least one commodity p, we define the *inefficiency ratio* as

$$I_{ij} = \frac{\sum_{p \in P} c_{ij} x_{ij}^p + f_{ij}}{\sum_{p \in P} x_{ij}^p}, \tag{6}$$

which is a measure of the average cost per unit of flow that is sent along arc (i,j). The lower the value of I_{ij}, the more efficient we regard arc (i,j) for accommodating flows. The average inefficiency ratio is defined as $\bar{I} = \sum_{(i,j) \in A} I_{ij} y_{ij} / \sum_{(i,j) \in A} y_{ij}$, and we define a set of *inefficient arcs* as $\mathcal{A}_I = \{(i,j) | y_{ij} = 1, I_{ij} > \bar{I}\}$, so that $(i,j) \in \mathcal{A}_I$ if arc (i,j) has an inefficiency ratio that is higher than the average. Our aim is to create neighbourhood moves that remove flows from some of the inefficient arcs in set $\mathcal{A}_I$.

We now describe how our *inefficient chains* are constructed from a subset of the inefficient arcs. First, an arc is randomly chosen from the set $\mathcal{A}_I$ of inefficient arcs to form a component of the first inefficient chain. If the current partial inefficient chain extends from node i to node j, then an arc $(h,i) \in \mathcal{A}_I$ or $(j,k) \in \mathcal{A}_I$ is added to the current chain (where nodes h and k are not included in the current chain). The arc added is chosen such that it has an inefficiency ratio that is as large as possible. Whenever an arc is included in a chain, it is deleted from $\mathcal{A}_I$. The process of extending the current chain continues until no further extension is possible. Unless $\mathcal{A}_I$ is empty or contains a single arc, the process iterates with a random arc chosen to start a new chain. When the process ends, any chains containing a single arc are discarded. The latter are likely to be included in inefficient chains in a subsequent ILS iteration, since inefficient chains are reconstructed from scratch at each ILS iteration.

Having constructed a set of inefficient chains, we now describe how our neighbourhood is formed. Each neighbour is based on a sub-chain of an inefficient chain and is defined by the starting node i and the ending node j of the sub-chain. If a chain comprises nodes $n_1 - n_2 - \cdots - n_m$, then the (i,j) values are considered in the order $(n_1,n_2),(n_1,n_3),\ldots,(n_1,n_m),(n_2,n_3),\ (n_2,n_4),\ldots,(n_2,n_m),\ldots,(n_{m-1},n_m)$. On the basis of our initial computational tests, we restrict our attention to sub-chains between i

and j comprising at most ζ arcs, which helps to reduce computation times but, at the same time, does not significantly restrict the diversity of potential neighbourhood moves.

The key aspect of our neighbourhood is the re-routing of flow from arcs of the sub-chain to other arcs of the network. An initial random decision is made as to whether a *full re-routing* or a *partial re-routing* is to be attempted for this sub-chain, with an equal probability for each choice. First, a list $\mathcal{P}_I$ of commodities is formed that have a positive flow through at least one arc of the sub-chain. To obtain a neighbour solution, the list of commodities is scanned and a re-routing of flow is attempted for each commodity p of $\mathcal{P}_I$ in turn. Suppose that the flow enters the sub-chain at node i^p, leaves the sub-chain at node j^p and the amount of flow is w^p.

Dijkstra's algorithm is applied to find a shortest path from node i^p to node j^p with the goal of finding a suitable path for the re-routing of flow. The shortest path problem is created according to the description given in Section 3.3, but with $\bar{c}_{ij} = \infty$ for each arc between i^p and j^p in the selected chain. For the case of full re-routing, the method for single path routing of Section 3.3 is used, while for partial re-routing the multiple-path routing method is used. If the resulting shortest path length is not finite, then the flow remains unchanged in the trail solution being constructed. Otherwise the flow is augmented as described in Section 3.3, and a corresponding reduction is made to the flows in the sub-chain. When all of the commodities of $\mathcal{P}_I$ are considered, the trial solution is a potential candidate for being selected as the neighbour defining the next move. Additional trial solutions are created by removing the first element of list $\mathcal{P}_I$ and repeating the process, again starting with a random decision as to whether a full or partial re-routing is to be attempted, until $\mathcal{P}_I$ is empty. The completed procedure is executed for every possible sub-chain.

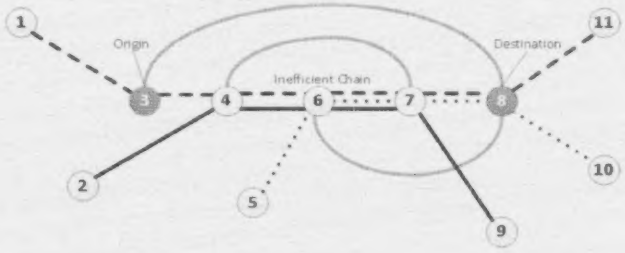


Figure 1: A typical inefficient chain and flow re-routing

We illustrate the idea of re-routing flows by an example shown in Figure 1. The example shows three commodities each with a different line pattern, and a graph where origin node 3 and destination node 8 define a part of the inefficient chain. The re-routing of the flows between nodes 3 and 8 causes individual commodity flow disconnections. The flow re-routings take place independently for each different commodity between its origin and destination nodes, i.e., the commodity shown with the solid black line must travel from node 4 to node 7, the dotted one must travel from node 6 to node 8, and the

dashed one from node 3 to node 8. The grey lines depict possible alternative re-routing paths within the network. All three flow re-routings, for this particular example, result in one single neighbour.

Another important component of our ILS is a frequency-based memory feature adopted by Paraskevopoulos et al. (2012) that penalizes potential moves that alter flows that have been changed frequently in previous iterations of the search. A vector $\vec{g}$ of size $|\mathcal{A}|$ is used to store each value g_{ij}, which is the number of times that the value of x_{ij}^p is changed for some $p \in \mathcal{P}$. After an improvement in the current solution is observed, $\vec{g}$ is reinitialized to the zero vector.

The following equation defines the local move cost from solution s to a trial solution s' as

$$\Delta f_{\text{move}}(s, s') = f(s') - f(s) + \beta \sum_{(i,j) \in A} b_{ij} g_{ij}, \tag{7}$$

where β is a scaling parameter, and b_{ij} has a value equal to 1 if the arc (i, j) participates in the current local move from s to s', and a value 0 otherwise. The component $\beta \sum_{(i,j) \in A} b_{ij} g_{ij}$ is added to the cost of the local move to penalize moves that involve frequently selected arcs.

Trial solutions with smaller values of Δf_{move} are generally preferred. However, it may be that this number is large enough to prevent the search from selecting a high-quality neighbour s'. To avert such cases, an aspiration criterion is used: if $f(s') < f(s_{\text{ILSbest}})$, the penalty component is ignored so that $\Delta f_{\text{move}} = f(s') - f(s)$. The neighbourhood search procedure is shown in Algorithm 3.

Algorithm 3: neighbourhood Search

Input: s' (current solution), M a large number
Output: s'' (best neighbour)
min = M;
for *All inefficient chains k of s' and for all combinations of nodes i, j in k* **do**
  $\mathcal{P}_I \leftarrow$ IdentifyDifferentCommodities(k, i, j);
  while $\mathcal{P}_I$ *is not empty* **do**
    if *isFeasible*$(k, i, j, \mathcal{P}_I)$ **then**
      $s^* \leftarrow$ CreateNeighbour$(k, i, j, \mathcal{P}_I)$;
    else
      RemoveFirstElement$(\mathcal{P}_I)$;Continue;
    if $\Delta f_{\text{move}}(s, s^*) <$ min **then**
      $s'' \leftarrow s^*$; min = $\Delta \text{f}_{\text{move}}(s, s^*)$;
    RemoveFirstElement$(\mathcal{P}_I)$;

In Algorithm 3 the function *IdentifyDifferentCommodities* forms the list $\mathcal{P}_I$ by identi-

fying the different commodities that have positive flows between the nodes i and j of an inefficient chain k. *CreateNeighbour* creates a neighbouring solution of s', and *RemoveFirstElement* removes the first element of the list. Finally, the *isFeasible* is a boolean function that returns "true" if a particular combination (k, i, j) leads to some re-routing of flow.

3.6.2 Ejection Cycles

A major component of the ILS is its perturbation strategy (Lourenço, Martin and Stützle 2002). The goal is to partially rebuild the current local optimum solution, such that the new diversified solution preserves some information from the local optimum. The proposed perturbation strategy in this paper, namely Ejection Cycles (EC), applies multiple cycle-based moves in the spirit of ejection chains (Glover 1996). The main idea of the ejection-chains strategy is to apply a compound move consisting of a series of consecutive local moves. Adopting this idea, our EC comprise a series of consecutive cycle moves of the type described in Section 3.6.1. The aim of EC is to perturb the structure of the current solution to achieve diversification, and also to remove some of the inefficient arcs from the solution.

The are two phases to creating the sequence of local moves. The first phase creates inefficient chains to re-route flow using similar ideas to Section 3.6.1, but considers the previous usage of arcs in local moves instead of cost and also allows flows in arcs to violate capacity constraints. The second phase attempts to remove infeasibility by doing further flow re-routing, again using arc usage in determining the path.

We now present more precise details of how our sequence of local moves is determined. In the first phase, we first find a set of inefficient chains and focus on sub-chains containing at most ζ arcs. For a given sub-chain, the list $\mathcal{P}_I$ is formed, and i^p, j^p and w^p are computed. The list of commodities is scanned and a re-routing of flow is performed for each commodity p of $\mathcal{P}_I$ in turn. However, in this re-routing, feasibility with respect to arc capacities is not enforced, as the second phase essentially operates a repair mechanism to restore feasibility. The first phase employs a full re-routing by applying Dijkstra's algorithm to find a shortest path from i^p to j^p, where cost for each arc $(i, j) \in \mathcal{A}$ is

$$\bar{c}_{ij} = \begin{cases} c_{ij}h_{ij} + f_{ij}(1 - \bar{y}_{ij}) & \text{if } (i,j) \notin \mathcal{F}, \\ \infty & \text{otherwise,} \end{cases} \tag{8}$$

where $h_{ij} - 1$ is the number of times that arc (i, j) has participated in a local move, and initialization sets $h_{ij} = 1$ for all $(i, j) \in \mathcal{A}$, and $\mathcal{F}$ is a set of forbidden arcs that initially comprises all arcs between i^p and j^p in the subchain. The h_{ij} values have a similar purpose to the g_{ij} values of Section 3.6.1 except that the method of initialization

is different. Also, the re-initialisation for q_{ij} is replaced by a scaling process for the h_{ij}. Specifically, to avoid h_{ij} become very large for some arcs (i,j), we periodically divide h_{ij} by $h_{\min}$ for all $(i,j) \in \mathcal{A}$, where $h_{\min} = \min_{(i,j)\in\mathcal{A}} h_{ij}$. If Dijkstra's algorithm returns a shortest path length of infinity, then the current sub-chain is not considered further and another one is selected. Otherwise, a flow of value w_p is added to each arc of the shortest path in the perturbed solution and removed from each arc of the sub-chain.

When re-routing of flow between nodes i^p and j^p of the sub-chain is complete for each $p \in \mathcal{P}_I$, we check if any arc has a flow that violates its capacity constraint. If there is no violation, then a new feasible solution is found and the EC terminates with a perturbed solution. When some flows violate arc capacities, we proceed as follows. Let $\mathcal{A}_V$ denote the set of arcs having a capacity violation. For all arcs $(i,j) \in \mathcal{A}_V$, a set of commodities $\mathcal{P}'_I$ is selected whose removal from (i,j) restores feasibility but keeps the capacity utilisation of the arc as high as possible. Specifically, the process of repeatedly selecting a commodity p with the largest flow x^p_{ij} in (i,j) is inserted in $\mathcal{P}'_I$ and the flow in (i,j) is reduced by x^p_{ij} is applied until the flow in (i,j) is reduced to exactly u_{ij} or the next selection would cause the flow in (i,j) to become strictly less than u_{ij}. In the latter case, the final commodity p selected for insertion into $\mathcal{P}'_I$ is chosen to have minimal flow in (i,j) from among those commodities where the removal of their flow from (i,j) reduces the total flow in (i,j) to be less than or equal to u_{ij}. Having formed $\mathcal{P}'_I$, the respective i^p, j^p and u^p are computed, and *infeasibility chains* that are formed in the same way as for inefficient chains, as described in Section 3.6.1.

Having formed the infeasibility chains, the aim is to re-route the flow in the chain using the method described above. More precisely, Dijkstra's algorithm to find a shortest path from the the starting node i^p of the sub-chain to the ending node j^p, where all arcs between i^p and j^p of the sub-chain are added to the set $\mathcal{F}$ and costs for the shortest path problem are defined by (8). If a suitable path for re-routing is found, then the trial solution s^* is updated. The process of re-routing flow in other infeasibility chains continues until no capacity violations occur or no further re-routing is possible due to the constraints imposed by set $\mathcal{F}$. If the former case, the EC terminates with a perturbed solution. In the latter case, the EC returns to the initial feasible solution s', the first commodity of set $\mathcal{P}_I$ is deleted and EC is applied on the remaining commodities in the set. As in Section 3.6.1, additional trial solutions are created by removing the first element of list $\mathcal{P}_I$ and repeating the process until $\mathcal{P}_I$ is empty. The complete procedure is applied to all sub-chains, and terminates when the first feasible perturbed solution is found. The pseudo code of the EC is given in Algorithm 4.

IdentifyViolatedArcs identifies the set of violated arcs $\mathcal{A}_V$. The function *NeighbourExists* is a boolean function that returns "true" if there exist an alternative path that the flow can be re-routed, regardless the capacity constraints at arcs. If no alternative paths are found (in case all neighbouring arcs have been assigned a cost of infinity), then *NeighbourExists* returns "false". The function $UpdateViolatedArcs$ identifies which of the arcs

Algorithm 4: Ejection Cycles

Input: s' (current solution)
Output: s^* (best neighbour)
for *All inefficient chains k of s' and for all combinations of nodes i, j* **do**
  $\mathcal{P}_I \leftarrow$ IdentifyDifferentCommodities(k, i, j);
  while $\mathcal{P}_I \neq \emptyset$ **do**
    ****First EC Iteration****
    $\mathcal{A}_V \leftarrow$ IdentifyViolatedArcs(k, i, j);
    if $\mathcal{A}_V = \emptyset$ **then**
      EndAlgorithm;
    else
      $\mathcal{P}'_I \leftarrow$ IdentifyExcessCommodities$(\mathcal{A}_V)$;
    ****Next EC Iterations****
    while $\mathcal{P}'_I \neq \emptyset$ **do**
      if *NeighbourExists*$(\mathcal{P}'_I)$ **then**
        $s^* \leftarrow$ CreateNeighbour$'(\mathcal{P}'_I)$;
        $\mathcal{A}_V \leftarrow$ UpdateViolatedArcs(k, i, j);
        if $\mathcal{A}_V = \emptyset$ **then**
          EndAlgorithm;
        else
          $\mathcal{P}'_I \leftarrow$ UpdateExcessCommodities$(\mathcal{A}_V)$;
      else
        RemoveFirstElement$(\mathcal{P}_I)$; $\mathcal{P}'_I \leftarrow \emptyset$;

of the re-routed paths are violated in terms of capacity constraints and updates the set of violated arcs $\mathcal{A}_V$. The function *IdentifyExcessCommodities* identifies the excess commodities that need to be removed from the violated arcs to restore capacity feasibility, while similarly *UpdateExcessCommodities* updates the excess commodities in the next iterations.

4 Computational Results

This section presents the computational analyses conducted to evaluate the performance of the proposed algorithm and comparisons with the state-of-the-art. The section is structured as follows: In Section 4.1, we describe the data sets used in the experiments, followed by Section 4.2 which explains the way that the algorithm parameters are calibrated. Sections 4.3 and 4.4 look at the effect of the network efficiency and the various strategies used on the performance of the algorithm. The way in which the components

of the proposed algorithm effect the solution quality is tested in Section 4.5. Finally, Section 4.6 presents extensive comparison results with state-of-the-art algorithms that have been proposed for the problem.

4.1 Data sets

To evaluate the performance of the proposed algorithm, computational experiments are conducted on the C and C+ benchmark instances described in Crainic et al. (2000). These sets include instances with 25, 30 and 100 nodes, 10 to 400 commodities and 100 to 700 arcs, and have been widely used in the literature. These instances differ from one another with respect to the nature of the arc capacities, which are either loose (L) or tight (T), and with respect to the relative importance of fixed costs (F) and the variable flow costs (V) per unit of flow. There also exist benchmark instances described by Alvarez et al. (2005) defined on an undirected graph using edges as opposed to a directed graph using arcs. These define a different problem than the one we address in this paper, as is discussed by Crainic et al. (2000), and is the reason why this set is not considered here.

The proposed algorithm was implemented in a Visual Studio 2010 environment using the C++ programming language, and all runs were performed on a single core Xeon E5507 2.27 GHz using CPLEX 12.1 as the optimizer.

4.2 Calibration

The algorithm uses five parameters; the number λ of initial solutions examined to produce the Reference Set R, the cardinality μ of R, the cardinality κ of CS, the maximum number δ of local search iterations without an improvement in the solution quality, and the maximum number $\vartheta_{\max}$ of CPLEX calls for which an improvement in the current solution is not observed. The termination criterion is the computational time. Most of the best solutions were derived in the first two hours, although our algorithm was allowed to run for up to 20000 seconds.

The scaling parameters α and β are self-calculated during the solution process, and are equal to the average cost of an arc in the current best solution found, i.e., $\alpha{=}\beta{=}f(s_{\text{best}})/\sum_{(i,j)\in A} y_{ij}^{s_{\text{best}}}$. The parameter λ does not appear to have a significant impact to the quality of the solutions; however, to have an adequate initial population size, we set it to 1500. Another parameter that seemed not to have significant impact is the size ζ of the sub-chains where local search takes place (see Section 3.6.1 for details). Parameter ζ was set equal to 4, which means that local search attempts to re-route the flows of a maximum of four arcs of a sub-chain. Larger values led to infeasibilities in the neighbouring solutions, either in the connectivity of the paths or the capacity of arcs.

We set $\kappa = 3$ to preserve the SS character of the proposed algorithm. Parameter κ needs to be larger than 2 to enhance the recombination process, but should be relatively small to ensure that a large number of possible combinations among the solutions of the Reference Set is considered. We tried 4 and 5 which resulted in a poor variety of offspring, due to the limited number of combinations. The latter problem was more prominent in the later SS iterations, when convergence is close and the need for different offspring is more apparent.

Parameters δ and $\vartheta_{\max}$ are interrelated as they typically control the total number of local search iterations. In particular, ϑ tracks CPLEX iterations; it is initialized to 1 and is incremented by one unit until $\vartheta_{\max}$ is reached. At each iteration, the number δ of local search iterations is set equal to 10ϑ. Initially, when ϑ is small, the local search resembles a steepest descent method, while as the search fails to improve the solution quality (close to large values of ϑ) more iterations are employed and "hill climbing" is typically enabled. Our experiments indicate that values of $\vartheta_{\max}$ equal to 6, 7, and 8 are appropriate, with values below 6 resulting in deterioration in the solution quality, and values greater than 8 slowing down the process without yielding any significant gain in the solution quality.

Table 1 shows the computational experiments conducted to investigate on the algorithm's behaviour with respect to different sets of parameters. Different parameter sets were used for different groups of problems. For large-scale problems, the Reference Set was of relatively small sizes and δ was assigned high values, whereas opposite settings were used for small to medium scale problems, for reasons described above. Table 1 shows the C and C+ benchmark instances classified into 6 groups according to their size. The label for each group is a vector depicting the number of nodes, the number of arcs and the number of commodities. The problem instances within each group differ in the tightness of the arc capacity constraints and the relative importance of the fixed costs and the costs of per unit of flow. The calibration was conducted by using one problem instance from each group, shown in the headings of the six main columns of Table 1. For each instance, ten runs, each with a run time of two hours, were conducted to retrieve the best solution values for each instance shown under the second column for each group. The parameter set that produces the best result for each group is fixed and used to solve the rest of the instances in that group, again with ten runs and each run with a time limit of 20000 seconds.

As Table 1 shows, the effect of the parameters of CEA varies according to the size of the problem solved. In small to medium scale instances, the evolutionary strategy had more impact than local search, since the cardinality of the neighbourhood is relatively small and local search is unable to adequately explore the search space. In contrast, the solution neighbourhood is enriched with more solutions and the impact of the local search is more significant in the solution process as the size of the problem instance increases. Driven by these observations, the size μ of the Reference Set takes larger values for small

Table 1: Calibration of the algorithm's parameters

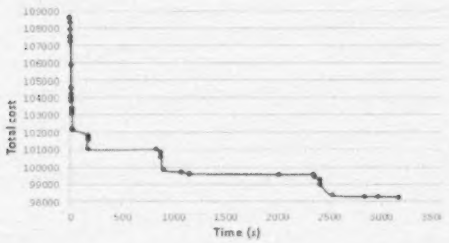
Group	**25-100-(10,30)**		**20-(230,300)-40**		**20-(230,300)-200**	
	$\vartheta_{\max}, \mu$	25-100-30FT	$\vartheta_{\max}, \mu$	20,230,40FT	$\vartheta_{\max}, \mu$	20,230,200VT
	6,30	86102	6,30	644180	6,30	98767
	6,40	85969	6,40	644995	6,40	98338
	6,80	85530	6,80	643187	6,80	98486
Parameter	7,30	85996	7,30	644413	7,30	98584
Sets	7,40	85948	7,40	644265	7,40	98209
	7,80	86089	7,80	643253	7,80	98945
	8,30	86059	8,30	644085	8,30	98451
	8,40	85932	8,40	643649	8,40	98767
	8,80	85535	8,80	643538	8,80	98807
Best	7,80	85530	7,80	643187	7,40	98209
Group	**100-400-(10,30)**		**30-(520,720)-100**		**30-(520,720)-400**	
	$\vartheta_{\max}, \mu$	100-400-30FT	$\vartheta_{\max}, \mu$	30,700,100FL	$\vartheta_{\max}, \mu$	30,700,400FT
	6,20	139661	6,20	61085	6,20	133822
	6,30	139805	6,30	61106	6,30	133889
	6,40	140503	6,40	61089	6,40	133861
Parameter	7,20	139995	7,20	60596	7,20	133245
Sets	7,30	140878	7,30	61054	7,30	133478
	7,40	140976	7,40	61188	7,40	133966
	8,20	140878	8,20	61284	8,20	133596
	8,30	139943	8,30	61266	8,30	133954
	8,40	140760	8,40	61176	8,40	133870
Best	7,40	139661	7,30	60596	7,30	133245

to medium scale problems, and relatively small values for the larger scale instances.

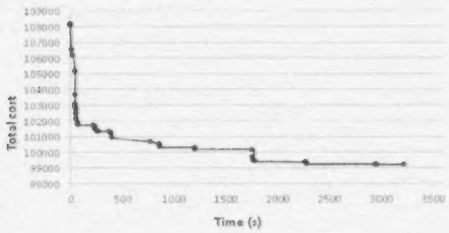
4.3 Network efficiency vs total cost

To illustrate the impact of the network efficiency on the solution cost, we have conducted analyses to shed light into the behaviour of the search on two problem instances, namely 20,230,200VT and 30,700,400VL. The network efficiency is defined with respect to either the maximum arc inefficiency or the average arc inefficiency, where the inefficiency measure is as defined in Section 3.6. The results are given in Figure 2, which shows how the two efficiency measures and the total cost change as the search progresses over time, separately for instance 20,230,200VT on the left and for instance 30,700,400VL on the right.

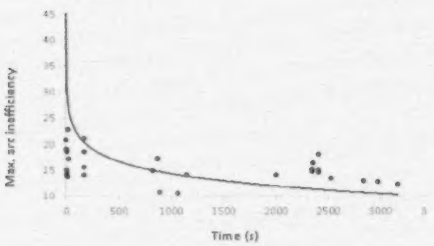
Figures 2(a) and 2(b) respectively show the changes observed in the value of the best solutions found for the 20,230,200VT and 30,700,400VL problem instances over time. Similarly, Figures 2(c) and 2(d) show the maximum inefficiency of an open arc for different solutions found over time. We observe that as the algorithm iterates, the maximum arc inefficiency is dramatically reduced and follows a logarithmic trend. Conversely, Figures 2(e) and 2(f) show an increase in the average efficiency of the arcs as the search progresses, which is indicative of an increase in the overall efficiency of the network as the solution quality is improved.



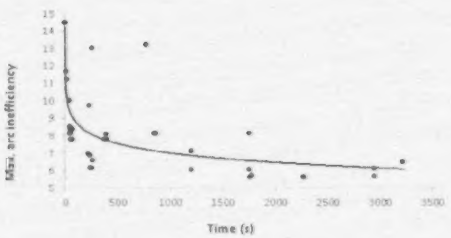
(a) Total cost vs time



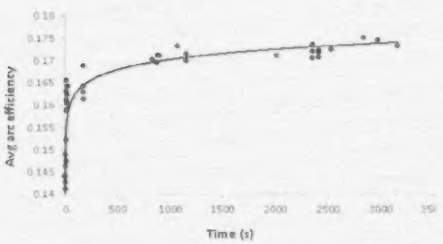
(b) Total cost vs time



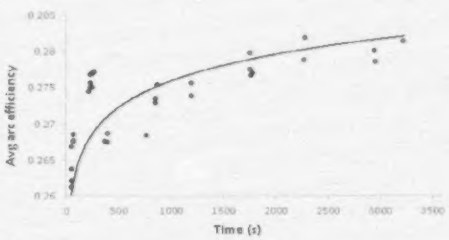
(c) Maximum arc inefficiency vs time



(d) Maximum arc inefficiency vs time



(e) Average arc efficiency vs time



(f) Average arc efficiency vs time

Figure 2: Arc efficiency and total cost tracked over time for instances 20,230,200VT (left) and 30,700,400VL (right)

4.4 Solvency Ratio vs random parent selection

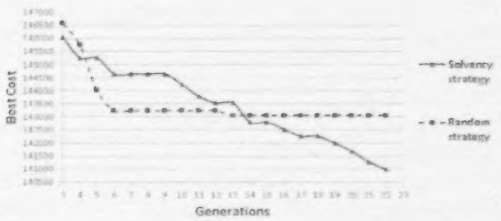
To illustrate the effectiveness of the Solvency Ratio, tests were conducted on two instances, namely 100-400-30-FT and 20,300,200FT, for the reason that these two instances typically present the general behaviour of the algorithm using solvency-based and random parent selection strategies.

Figure 3 presents the comparisons between the two strategies. The first two SS iterations are used as a warm up for the solvency strategy, which is enabled from the third SS iteration onwards as is apparent from the figures. Figures 3(a) and 3(b) show how the best solution values evolve over time. For 100,400,40FT, it is easily seen that solutions obtained by the random-based strategy are quickly trapped in a local optimum, whereas the solvency-based strategy is slower to improve the best solution initially, but displays a gradual yet continual reduction in the overall cost as the generations evolve, and terminates with a better overall solution. Instance 20,300,200FT exhibits a similar pattern, i.e., the solvency strategy provides a large improvement in the early SS iterations and then follows a less steep drop as the algorithm continues to improve the total cost. The random strategy is again trapped in a local optimum at iteration 12. Figures 3(c) and 3(d) show the changes in the average solution cost in the Reference Set over the SS iterations. The main observations on the behaviour of the solvency-bases strategy are similar to the first two figures.

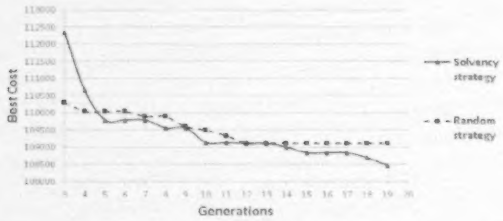
A "healthy" evolutionary process should typically produce a decent number of educated offspring at each SS iteration. Figures 4(e) and 4(f) show that this is also the case in the proposed algorithm. In particular, the figures show that the random strategy has difficulties in producing educated offspring and therefore results in premature convergence. In contrast, the solvency strategy is able to update the Reference Set with educated offspring even near termination.

4.5 The impact of the CEA's main components on the solution quality

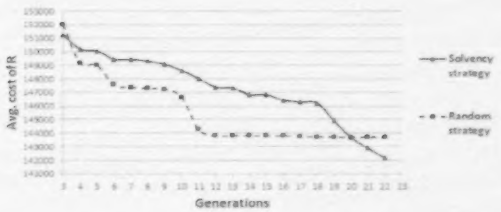
Experimentation was conducted on different versions of the proposed CEA to investigate the effect of various components on the final solution quality. Three versions of CEA were thus considered: (i) Version "\EC" is where a random perturbation strategy is used instead of EC. According to this random strategy, 25% of the commodities are selected at random, which are then removed and re-routed via the construction mechanism as discussed in Section 3.4. (ii) Version"\SolvR" replaces the Solvency ratio strategy with a random strategy for the parent selection and the Reference Set updating criteria. According to the random strategy, the parents that comprise the Candidate Set are selected at random and the elitist updating criteria described in 3.5.1 are used. (iii) Version "\Ineff",



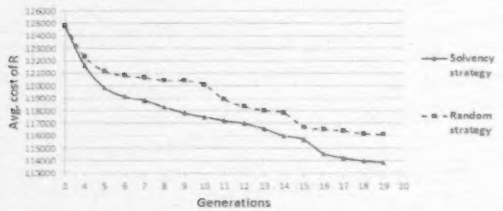
(a) Total cost: 100-400-30FT



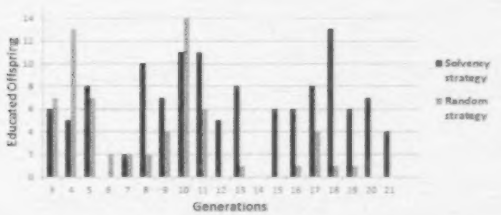
(b) Total cost: 20,300,200FT



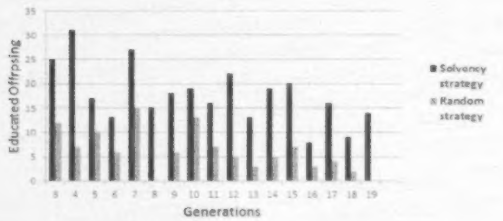
(c) Average cost for R: 100-400-30FT



(d) Average cost for R: 20,300,200FT



(e) Educated offspring: 100-400-30FT



(f) Educated offspring: 20-300-200FT

Figure 3: Solvency-based vs random-based strategies for instances 100-400-30FT (left) and 20,300,200FT (right)

Table 2: Results from different versions of CEA on the benchmark instances of Crainic et al. (2000)

Instances	CEA	% Deviations			Instances	CEA	% Deviations		
		\EC	\SolvR	\Ineff			\EC	\SolvR	\Ineff
25,100,10VL	14712	0.00	0.00	0.00	100,400,10FL	23949	−1.23	−0.98	−3.54
25,100,10FL	14941	0.00	−0.64	−1.91	100,400,10FT	66240	−6.34	−4.87	−10.79
25,100,10FT	49899	0.00	−1.78	−2.08	100,400,30VT	385163	−0.20	−0.83	−1.05
25,100,30VT	365272	0.00	−1.09	0.00	100,400,30FL	49577	−3.01	−3.06	−4.48
25,100,30FL	37324	−0.98	−4.47	−3.38	100,400,30FT	139661	−1.78	−3.64	−2.33
25,100,30FT	85530	−0.65	−0.84	−0.79	30,520,100VL	54109	−0.99	−1.01	−0.99
20,230,40VL	423848	−0.05	−0.19	−0.19	30,520,100FL	95302	−0.14	−2.13	−2.76
20,230,40VT	371475	−1.18	−0.09	−0.09	30,520,100VT	52284	−0.49	−1.30	−1.32
20,230,40FT	643187	−0.22	−0.33	−0.30	30,520,100FT	98525	−0.64	−1.76	−3.58
20,300,40VL	429398	−0.29	0.00	0.00	30,700,100VL	47619	−0.60	−0.84	−0.64
20,300,40FL	586077	−0.49	−0.99	−0.76	30,700,100FL	60596	−1.27	−3.38	−3.32
20,300,40VT	464509	−0.53	−0.32	0.00	30,700,100VT	46084	−1.08	−1.25	−1.64
20,300,40FT	604198	−0.28	−0.01	0.00	30,700,100FT	55271	−1.24	−1.92	−2.50
20,230,200VL	94468	−0.75	−1.23	−4.76	30,520,400VL	113694	−1.42	−1.06	−1.50
20,230,200FL	139002	−1.89	−0.96	−6.13	30,520,400FL	151688	−1.87	−1.10	−5.04
20,230,200VT	98209	−1.10	−0.66	−4.60	30,520,400VT	116322	−0.70	−1.24	−1.86
20,230,200FT	137131	−3.62	−0.46	−5.69	30,520,400FT	154425	−1.90	−2.58	−2.74
20,300,200VL	75288	−1.21	−0.98	−4.03	30,700,400VL	99222	−0.68	−1.19	−2.36
20,300,200FL	117320	−0.84	−0.04	−2.25	30,700,400FL	137112	−2.05	−4.12	−6.26
20,300,200VT	75607	−0.99	−0.93	−1.65	30,700,400VT	96388	−1.48	−1.42	−2.54
20,300,200FT	108459	−0.28	−2.60	−6.28	30,700,400FT	133245	−1.08	−0.32	−2.43
100,400,10VL	28426	−0.18	−1.31	−0.55					

performs local moves on chains composed by all arcs of the network, disregarding any preference given to inefficient chains.

The results of the experiments are reported in Table 2. In this table, column "CEA" shows the best results of these experiments, derived from 10 runs for each problem instance, where the computational time of each run is limited to two hours. The values under column "% Deviations" in Table 2 show the percent deviations of the solution values obtained by the three versions of the CEA from those of the best solution value. In particular, the deviations are calculated as $100(v(\text{CEA}) - v(\text{Alg}))/v(\text{CEA})$, where $v(\text{Alg})$ is the solution value obtained by one of the three versions of CEA.

From Table 2, it can be easily observed that the impact of the EC in the quality of the final solution is significant, and can yield reductions of up to 6.34% in total cost. The maximum improvements afforded by the Solvency Ratio and the Inefficiency Measures are 4.87% and 10.79%, respectively. A negative deviation value in this table indicates that the solution found by the CEA is better. On average, the most significant impact comes from the Inefficiency Measures component with an average deviation of −2.54%. The same statistics for the Solvency Ratio and the EC are −1.39% and −1.06%, respectively.

4.6 Comparative analysis

In this section, we report comparative computational results of the proposed algorithm with the Cycle-based Tabu Search (CTS) of Ghamlouche et al. (2003), Path Relinking (PR) by Ghamlouche et al. (2004), Multilevel Cooperative Algorithm (MCA) by Crainic et al. (2006), Capacity Scaling Heuristic (CSH) by Katayama et al. (2009), IP Search (IPS) by Hewitt et al. (2010), the two algorithms based on Simulated Annealing and Column Generation (SACG1 and SACG2) described by Yaghini et al. (2013) the results for which are reported with time limits 600 and a 18000 seconds, respectively, and Local Branching (LocalB) by Rodríguez-Martín and Salazar-González (2010). The algorithm described by Alvarez et al. (2005) could not be included in the comparisons as the authors do not report any results with the instances tested here; instead they use their own benchmark instances. The reason for not being able to test our algorithm on the Alvarez et al. (2005) benchmark set is that these instances are based on an undirected graph and work with edges, whereas the problem we solve is on a directed graph and our algorithm has been developed to operate on arcs.

Table 3 shows the comparison results where the first column shows the name of the instance as characterized by the number of nodes, the number of arcs and the number of commodities. The solution values obtained by the proposed algorithm are reported under column "CEA". The remaining five columns report the relative percentage deviations of the solution values found by the CEA from those reported by the papers quoted above, and is calculated as $100(v(\text{CEA}) - v(\text{Alg}))/v(\text{CEA})$, where $v(\text{Alg})$ indicates the solution value produced by the corresponding algorithm and $v(\text{CEA})$ the solution value produced by the CEA. A negative value indicates that the solution found by the CEA is better.

The first seven rows describe, to the best that we were able to extract, the computational resources used to run the algorithms. The row titled "T.Lim.(sec)" reports the time limit used by the authors of the corresponding algorithm, whereas the "Used Cores" row indicates how many cores from the original configuration of the CPU were used to run the algorithm. It is assumed that the computational power increases linearly with the number of cores used. Due to different computing facilities, we have normalized the computational times using the approach described in Dongarra (2013) and data from http://www.cpubenchmark.net/. All comparisons were made according to the Passmark CPU Score (PCPUS). As we were unable to find PCPUS for Sun systems on http://www.cpubenchmark.net/, we used the Dongarra (2013) list, and selected an Intel equivalent. The final scores are reported in the row titled "PCPU Score". The running times were normalized by using CEA as the reference point, i.e., Norm.TL(Alg)= PCPUS(Alg)TL(Alg)/PCPUS(CEA).

The table also reports some summary statistics in the last six rows, including the median and the average of the deviations. The p-value is the result of a T-test with a double tail conducted to test whether the differences between the results are statistically

significant. The "MaxImpr." row shows the maximum improvement afforded by the CEA. The lower this value is, the better the performance of the algorithm. The LeastGap row shows the maximum deviation over instances for which CEA did not find a better solution. Finally, the row named "Impr./43" shows the number of instances out of the total 43 tested, where CEA yielded the same or better results over the algorithm it is compared with.

As the results shown in Table 3 indicate, the CEA is competitive with the state-of-the-art. In particular, CEA is able to produce optimal solutions for the 25,100,10, 20,300,40 sets of instances as well as for the large scale problem instances 100,400,10FL and 30,700,100VL. Furthermore, the CEA finds optimal solutions for 25,100,30FT and 20,230,40VL which could not be found by any of the heuristics used for comparisons with the exception of the ones described by Yaghini et al. (2013). The maximum deviations of the CEA are -8.75% compared with CTS, -8.46% compared with PR, -5.32% compared with MCA, -12.21% compared with CSH, -11.28% compared with IPS,-17.07% and -1.06% compared with SACG1 and SACG2, respectively, and -23.81% compared with LocalB.

Noteworthy is the fact that on large-scale problem instances 20,300,200FT, 100,400,30FT and 30,520,100FT, new best solutions were obtained with values 107546, 139535 and 97856, respectively. These instances have up to 100 nodes, 520 arcs and 200 commodities, and the new best solutions deviate by -0.29%, -1.06% and -0.70% over the previous best known solutions, respectively.

On average measures, CEA outperforms CTS, PR and MCA by achieving average improvements of -3.49%, -3.11%, and -2.44%, respectively. Compared with the rest, the CEA still remains competitive with average deviations sitting at -0.37% from CHS, -0.72% from IPS, -1.07% from SACG1, 0.15% from SACG2 and -1.81% from LocalB. The improvements are statistically significant as the p-value indicates (with a reference of 0.05) over CTS, PR, MCA, IPS, SACG1 and LocalB. Compared with CHS, the proposed algorithm produces better results by -0.37% on average, however the improvement is not statistically significant as the p-Value is equal to $0.28(>0.05)$. Similarly, although SACG2 produces results that are better by 0.15% the difference is not statistically significant as the p-Value is equal to 0.75. We also note that we are unable to consider the result of SACG2 for instance 30,520,400FT as this value is lower than the lower bound 150009 reported by Katayama et al. (2009), and any comparison for this instance would therefore be misleading.

The above comparisons are based on the results derived by using the running time limits imposed by the original authors. Even though our time limit was 20000 sec, the CEA was able to discover the best solution in less than two hours for most problem instances. For very large-scale instances, improvements were observed in later SS iterations which necessitated additional running time. The latter observation is as one would

Table 3: Comparisons to the state-of-the-art algorithms on the C and C+ sets of benchmark instances of Crainic et al. (2000)

Instances	OPT/LB	CEA	% Deviation							
			CTS	PR	MCA	SCH	IPS	SACG1	SACG2	LocalB
T.Lim(sec)		20000	36000	36000	18000	2610	900	600	18000	600
CPU		Xeon E5507	Sun U. 60/2300	Sun En. 10000	Sun En. 10000	Pentium E5800	8xXeon	Core2Duo E6850	Core2Duo E6850	Core2Duo E4600
GHz		2.27	0.4	0.4	0.4	3.2	2.66	3.00	3.00	2.4
Used Cores		1	1	1.00	64	2	8x1	2	2	2
PCPU Score		3212/4	238	145	64x238	1664	435x8	1985	1985	1379
Norm. TL		20000	2353	1434	45877	5408	3900	1483	44436	1030
25,100,10VL	14712 O	14712	0.00	0.00	0.00	0.00	n/a	0.00	0.00	0.00
25,100,10FL	14941 O	14941	0.00	0.00	0.00	−0.64	n/a	0.00	0.00	0.00
25,100,10FT	49899 O	49899	0.00	0.00	−0.08	−1.75	n/a	0.00	0.00	0.00
25,100,30VT	365272 O	365272	−0.03	−0.03	−0.03	0.00	n/a	0.00	0.00	0.00
25,100,30FL	37055 O	37324	−0.69	−0.88	−0.76	−0.39	n/a	0.71	0.71	−0.01
25,100,30FT	85530 O	85530	−0.90	−1.05	−1.09	−0.32	n/a	0.00	0.00	0.00
20,230,40VL	423848 O	423848	−0.22	−0.13	−0.67	−0.05	−0.13	0.00	0.00	0.00
20,230,40VT	371475 O	371475	−0.11	−0.09	0.00	−0.12	−0.08	0.00	0.00	0.00
20,230,40FT	643036 O	643187	−0.41	−0.37	−1.51	−0.20	0.00	0.02	0.02	0.02
20,300,40VL	429398 O	429398	−0.03	0.00	−0.10	0.00	0.00	0.00	0.00	0.00
20,300,40FL	586077 O	586077	−1.24	−0.74	−1.27	−0.29	0.00	0.00	0.00	0.00
20,300,40VT	464509 O	464509	−0.05	0.00	−0.32	−0.01	0.00	−0.03	−0.03	0.00
20,300,40FT	604198 O	604198	−0.48	−0.96	−2.48	0.00	0.00	0.00	0.00	0.00
20,230,200VL	92598 L	94468	−4.79	−6.28	−4.35	0.23	−0.67	−0.88	0.20	−0.88
20,230,200FL	133512 L	138954	−5.46	−6.50	−3.02	0.64	−1.65	−0.58	0.80	−3.23
20,230,200VT	97344 L	98209	−6.66	−6.60	−3.89	0.25	−1.22	0.23	0.30	0.17
20,230,200FT	132432 L	137131	−7.48	−7.60	−2.96	0.73	−2.29	−2.65	0.34	−2.91
20,300,200VL	73759 L	75279	−7.36	−3.86	−3.89	0.49	−0.05	−0.95	0.50	−1.46
20,300,200FL	111655 L	116801	−5.60	−5.72	−4.41	0.87	−0.63	−0.98	0.32	−20.00
20,300,200VT	74991 O	75444	−5.53	−4.54	−2.40	0.19	−1.00	0.03	0.60	−0.96
20,300,200FT	104334 L	107546	−6.45	−5.61	−3.37	−0.29	−2.60	−2.01	−1.02	−2.10
100,400,10VL	28423 O	28423	−0.89	−0.22	−0.46	−0.01	0.00	0.00	0.00	0.00
100,400,10FL	23949 O	23949	0.00	−0.30	−0.30	−2.13	0.00	0.00	0.00	−3.09
100,400,10FT	59470 L	65563	−2.21	0.43	−1.10	−12.21	−0.49	0.60	0.50	−2.74
100,400,30VT	384560 L	384999	−0.13	0.02	−0.07	0.03	0.04	0.05	0.05	0.05
100,400,30FL	47459 L	49465	−4.22	−3.76	−2.00	−5.03	−0.46	−0.35	0.40	−0.82
100,400,30FT	127825 L	139535	−4.02	−1.31	−4.43	−3.42	−1.31	−1.44	−1.36	−1.50
30,520,100VL	53958 L	54099	−1.59	−1.49	−3.06	0.02	−0.03	0.21	0.21	0.13
30,520,100FL	91285 L	95142	−4.67	−7.26	−4.91	0.36	0.79	0.88	1.13	−1.17
30,520,100VT	51825 L	52182	−1.54	−1.60	−2.55	−0.19	0.02	−0.40	−0.12	0.10
30,520,100FT	94646 L	97855	−7.83	−8.46	−4.72	−1.00	−1.05	−2.38	−0.70	−3.32
30,700,100VL	47603 O	47603	−1.67	−2.35	−2.66	−0.07	−0.02	−0.83	0.00	0.00
30,700,100FL	58772 L	60538	−3.19	−4.22	−5.32	0.57	−0.27	−0.07	0.24	0.44
30,700,100VT	45552 L	46082	−2.05	−2.45	−2.98	−0.19	0.08	−0.14	0.27	0.38
30,700,100FT	54233 L	55135	−4.99	−2.61	−3.22	−0.41	−0.86	0.22	0.29	0.06
30,520,400VL	111992 L	113193	−6.59	−5.50	−2.19	0.31	−0.75	−0.89	−0.47	−1.04
30,520,400FL	146809 L	151145	−6.59	−7.92	−3.61	1.12	−2.03	−3.10	0.39	−4.35
30,520,400VT	114237 L	115697	−5.09	−3.87	−4.57	0.91	0.67	0.07	0.10	0.40
30,520,400FT	150009 L	154425	−8.75	−5.99	−3.75	1.09	−0.12	−3.59	n/a	−9.15
30,700,400VL	96741 L	98729	−8.15	−6.47	−3.95	0.77	0.01	−3.04	−0.59	−5.12
30,700,400FL	130724 L	137112	−8.63	−5.77	−5.01	1.49	−11.28	−17.07	2.29	−23.81
30,700,400VT	94118 L	96130	−5.77	−5.29	−3.19	0.86	−0.04	−0.95	0.52	−0.57
30,700,400FT	127666 L	132425	−7.82	−6.49	−4.41	1.72	0.60	−6.85	0.72	−9.44
		Median	−3.19	−2.45	−2.66	0.00	−0.08	0.00	0.03	0.00
		Avg	−3.49	−3.11	−2.44	−0.37	−0.72	−1.07	0.15	−1.81
		p−value	0.000	0.000	0.000	0.280	0.029	0.019	0.075	0.006
		MaxImpr.	−8.75	−8.46	−5.32	−12.21	−11.28	−17.07	−1.06	−23.81
		LeastGap	0.00	0.43	0.00	1.72	0.79	0.88	2.29	0.44
		Impr./43	43	41	43	24	30	33	20	34

Table 4: Computational times (in sec) and comparisons with the state-of-the-art

Instances	CTS	PR	MCA	CSH	SACG2	IP	CEA
CPU score	238	145	238x64	371	1985	3480	803
25,100,10VL	14.5	2.3	118.6	2.7	291.7	n/a	12.2
25,100,10FL	15.9	2.5	133.7	14.7	954.2	n/a	32.2
25,100,10FT	15.2	4.1	228.6	7.7	415.3	n/a	1240.0
25,100,30VT	66.3	18.3	961.7	6.6	6081.1	n/a	148.0
25,100,30FL	63.8	13.6	713.2	30.5	2719.2	n/a	1408.0
25,100,30FT	66.6	17.5	920	20.7	5811.6	n/a	55.0
20,230,40VL	109.8	26.9	1411.3	6.2	3087.5	151.7	1726.0
20,230,40VT	129.1	28.3	1488.1	6.8	487	39.0	142.0
20,230,40FT	125.5	31.1	1633.2	7.9	699.6	3523.3	1410.0
20,300,40VL	181.2	40.6	2133	6.6	274.4	1430.1	132.0
20,300,40FL	172.5	41.2	2165.3	12.8	454.8	3839.7	652.0
20,300,40VT	174.8	44.8	2351.2	7.9	7326.9	3848.4	6974.0
20,300,40FT	166.1	38.7	2033.5	9.1	2585.7	17.3	468.0
20,230,200VL	789.3	450.5	23662.7	916.1	9942.3	177.7	5242.0
20,230,200FL	805.7	519.7	27299	3435.8	6145.3	195.0	7805.0
20,230,200VT	760.4	399.2	20969.1	1084.8	3391.6	3562.3	4690.0
20,230,200FT	924.8	611.4	32112.4	4028.0	2294.0	2994.6	6246.0
20,300,200VL	1211.3	643.9	33821.5	720.5	22561.8	3558	7152.0
20,300,200FL	1294.6	724.6	38057.2	2673.0	2491.8	676.1	7042.0
20,300,200VT	1128.6	708.6	37218.8	888.4	31898.4	82.3	12351.3
20,300,200FT	1380.1	696.5	36582.1	3567.8	24054.8	125.7	11294.9
100,400,10VL	99.7	16.1	846	12	44495.6	104.0	130.0
100,400,10FL	90.9	15	786.3	193.3	23103.1	294.7	724.0
100,400,10FT	185.7	37.9	1990.8	115.2	14295.5	3475.7	5280.0
100,400,30VT	585.5	89	4673.9	36.9	9868.1	2973.0	2776.0
100,400,30FL	385.5	56.9	2987.6	1287.9	10384.8	1681.5	6890.0
100,400,30FT	554.2	86.8	4561.1	318.9	16918.2	1716.2	16683.2
30,520,100VL	994.7	215.6	11325.4	55.7	44495.6	944.8	19087.1
30,520,100FL	1195.2	263.6	13847.3	842.4	34563.2	979.4	1700.0
30,520,100VT	1031.8	273.3	14356.6	77.3	44495.6	1971.9	18058.5
30,520,100FT	1164	275	14441.9	413.8	35314.7	3532.0	17771.9
30,700,100VL	1303	336	17646.7	79.8	61.8	1707.5	14069.3
30,700,100FL	1409.3	331.8	17427.6	237.3	42	3250.3	16934.5
30,700,100VT	1351.6	342	17964.5	95.9	71.7	2691.3	7202.0
30,700,100FT	1442.3	308.1	16181.4	200.6	160.7	2019.5	15875.0
30,520,400VL	10827.3	4961.7	260607.6	1177	182.9	138.7	19865.6
30,520,400FL	12723.8	6621.5	347787.5	5409.3	496.9	3211.3	18390.1
30,520,400VT	8362.3	4169.3	218987	476.6	402.9	1607.8	19622.8
30,520,400FT	11858.8	9421.1	494833.2	3468.7	479.6	1677.2	7042.0
30,700,400VL	7355.4	4029.4	211643.2	983.9	417.8	962.1	10307.3
30,700,400FL	20610.9	13663	717638.7	3694.6	1058	3727.0	7202.0
30,700,400VT	10366.2	4385.9	230366.5	1552.3	2316.2	1581.8	15041.7
30,700,400FT	15376	8114.3	426196.3	3081.6	2983.7	975.1	19992.1
Avg	2764.4	1466.9	77049.1	959.7	9780.9	1768.8	7834.1

expect with evolutionary algorithms, i.e., a number of SS iterations are needed in order that the initial population of solutions can be evolved such that high quality solutions can be produced.

The final set of comparisons relate to the computation times needed by the CEA and other state-of-the-art algorithms to obtain the best solutions, which are shown in Table 4. The times for the latter group have been adjusted using the PCPUSs introduced in Table 3, such that an objective comparison can be made. As CEA outperforms CTS, PR and MCA in terms of the solution quality, the main focus will be on comparisons with CSH, SACG2 and IP. As the table shows, CEA needs 25% less running time than SACG2 on average, for producing solutions that deviate by 0.15% from the ones produced by SACG2. It is worth noting that the MCNDP is a planning problem at strategic/tactical level of decision making and it is not unreasonable to devote more time in solving it.

5 Conclusions and Further Research

This paper presented an evolutionary algorithm for the Fixed Charge Capacitated Multi-Commodity Network Design Problem. The proposed methodology evolves a pool of solutions using Scatter Search principles, and includes an Iterated Local Search as an improvement method. The latter introduces new cycle-based neighbourhood structures, short and long term memory structures for guiding the search, and an efficient perturbation strategy, inspired by Ejection Chains, to enable the search escape from local optima. An efficient recombination strategy is introduced which dynamically adjusts the preferences for inherited solutions based on the search history.

Computational experiments on the benchmark instances of Crainic et al. (2000) show that the proposed CEA is highly competitive compared to state-of-the-art approaches. In particular, CEA is able to reproduce the 13 out of 17 optimum solutions for 17 problem instances previously solved by exact algorithms. CEA was also able to produce three new best solutions, in large-scale problem instances. In general terms, CEA's performance is strong, thus placing it among the most efficient algorithms for the MCNDP.

In terms of further research, a promising research direction is the use of a knowledge base where favourable paths for the commodities would be stored not only for speeding up the algorithm but also for guiding the algorithm towards producing unexplored solution structures. Another direction is to look at decomposition techniques to solve the flow subproblems with a view to reducing the computational times. Finally, it is worthwhile to explore the proposed evolutionary algorithm for solving other variants of the MCNDP or even to other problems that share common features with MCNDP.

Acknowledgements

The authors gratefully acknowledge the following sources of funding for this project: the Engineering and Physical Sciences Research Council (EPSRC), the Rail Safety and Standards Board (RSSB) and Department of Transport, UK, the Natural Sciences and Engineering Council of Canada (NSERC) through its Discovery Grant programs, the Faculty Strategic Research Fund provided by the Faculty of Business and Law at the University of Southampton, Fonds de recherche du Québec, Canada for their infrastructure grants.

References

Alvarez, AM, JL González-Velarde, K De-Alba. 2005. Scatter search for network design problem. *Annals Oper. Res.* **138** 159–178.

Balakrishnan, A, TL Magnanti, P Mirchandani. 1997. Network design. F Dell'Amico, M Maffioli, S Martello, eds., *Bibliographies in Combinatorial Optimization*. John Wiley and Sons, 311–334.

Crainic, TG, M Gendreau, J Farvolden. 2000. A simplex-based tabu search for capacitated network design. *INFORMS J. Comput.* **12** 223–236.

Crainic, TG, A Frangioni, B Gendron. 2001. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Appl. Math.* **112** 73–99.

Crainic, TG, M Gendreau. 2002. Cooperative parallel tabu search for capacitated network design. *J. Heur.* **8** 601–627.

Crainic, TG, M Gendreau. 2007. A scatter search heuristic for the fixed-charged capacitated network design problem. Doerner, KF, M Gendreau, P Greistorfer, WJ Gutjahr, RF Hartl and M Reimann, ed., *Metaheuristics - Progress in Complex Systems Optimization*. Springer, 25–40.

Dongarra JJ. 2013. Performance of Various Computers Using Standard Linear Equations Software. *CS - 89 - 85, University of Manchester*.

Crainic, TG, Y Li, M Toulouse. 2006. A first multilevel cooperative algorithm for capacitated multicommodity network design. *Comput. Oper. Res.* **33** 2602–2622.

Feo, TA, MGC Resende. 1995. Greedy Randomized Adaptive Search Procedures. *J. Global Optim.* **6**(2) 109–133.

Fischetti M, Lodi A. 2003. Local branching. *Math. Programming* **98** 23–47.

Frangioni, A, B Gendron. 2001. 0–1 Reformulations of the multicommodity network design problem. *Discrete Appl. Math.* **112** 73–99.

Gendron B, TG Crainic, A Frangioni. 1998. Multicommodity capacitated network design. P Soriano, B Sanso, eds., *Telecommunications Network Planning*. Kluwer, 1–19.

Ghamlouche, I, TG Crainic, M Gendreau. 2003. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Oper. Res.* **51** 655–667.

Ghamlouche, I, TG Crainic, M Gendreau. 2004. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals Oper. Res.* **131** 109–134.

Glover, F. 1996. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Appl. Math.* **65** 223–253.

Hewitt M, GL Nemhauser, MWP Savelsbergh. 2010. Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS J. Comput.* **22**(2) 314–325.

Hewitt M, GL Nemhauser, MWP Savelsbergh. 2012. Branch and price guided search for integer programs with an application to the multicommodity fixed-charge network flow problem. *INFORMS J. Comput.* **Articles in advance** 1–15.

Holmberg, K, D Yuan. 2000. A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Oper. Res.* **48** 461–481.

Katayama, N, M Chen, M Kubo. 2009. A capacity scaling heuristic for the multicommodity capacitated network design problem. *J. Comput. Appl. Math.* **232** 90–101.

Lourenço, HR, O Martin, T Stützle. 2002. Iterated local search. Dorigo M Corne D, Glover F, eds., *Handbook of Metaheuristics: International Series in Operations Research & Management Science*. Kluwer, 321–353.

Magnanti, TL, RT Wong. 1986. Network design and transportation planning: models and algorithms. *Transportation Sci.* **1** 1–55.

Minoux, M. 1986. Network synthesis and optimum network design problems: models, solution methods and applications. *Networks* **19** 313–360.

Paraskevopoulos DC, CD Tarantilis, G Ioannou. 2012. Solving Project Scheduling Problems with Resource Constraints via an Event List-based Evolutionary Algorithm. *Expert Systems Appl.* **39**(4) 3983–3994.

Rodríguez-Martín I, Salazar-González JJ. 2010. A local branching heuristic for the capacitated fixed-charge network design problem. *Comput. Oper. Res.* **37**(3) 575–581.

Yaghini M, M Rahbar, M Karimi. 2013. A hybrid simulated annealing and column generation approach for capacitated multicommodity network design. *J. Oper. Res. Soc.* **64** 1010–1020.